1 核心算法

1.1 三次样条曲线

给定区间 $[a,b]$ 上 $n+1$ 个节点 $a=x_0<x_1<\cdots<x_n=b$ 和这些节点上的函数值 $\{f(x_i)=y_i, i=0,1,\cdots,n\}$，若 $S(x)$ 满足：$\{S(x_i)=y_i, i=0,1,\cdots,n\}$，$S(x)$ 在每个小区间 $[x_i,x_{i+1}]$ 上至多是一个三次多项式，且 $S(x)$ 在 $[a,b]$ 上有连续二阶导数，则称 $S(x)$ 为 $f(x)$ 关于剖分 $a=x_0<x_1<\cdots<x_n=b$ 的三次样条差值函数。

用插值点 $\{(x_i,S''(x_i)),(x_{i+1},S''(x_{i+1}))\}$ 作线性插值，记 $S''(x_i)=M_i, i=0,1,\cdots,n-1$，处理后可以得到：

$$\mu_i M_{i-1}+2M_i+\lambda_i M_{i+1}=d_i,\quad i=0,1,\cdots,n-1$$

其中

$$\lambda_i=\frac{h_i}{h_i+h_{i-1}},\quad \mu_i=1-\lambda_i$$

$$d_i=\frac{6}{h_i+h_{i-1}}\left(\frac{y_{i+1}-y_i}{h_i}-\frac{y_i-y_{i-1}}{h_{i-1}}\right)=6f(x_{i-1},x_i,x_{i+1}),\quad h_i=x_{i+1}-x_i$$

在选定自然边界条件：$M_0=M_n=0$ 时：

$$\begin{pmatrix} 2 & \lambda_1 & & & \\ \mu_2 & 2 & \lambda_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \mu_{n-2} & 2 & \lambda_{n-2} \\ & & & \mu_{n-1} & 2 \end{pmatrix}\begin{pmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-2} \\ M_{n-1}\end{pmatrix}=\begin{pmatrix} d_1-\mu_1 M_0 \\ d_2 \\ \vdots \\ d_{n-2} \\ d_{n-1}-\lambda_{n-1}M_n\end{pmatrix}$$

其他边界条件需要另行讨论，但是在本次实验中未涉及，因此不做赘述。具体算法详见 Algorithm 1, 2, 3，4。

2 实验结果

2.1 实验 (a)

```
S_i(x) = (-0.254)x^3 + (-6.864)x^2 + (-60.740)x + (-175.849), x in [-9.000, -8.000]
S_i(x) = (0.204)x^3 + (4.129)x^2 + (27.204)x + (58.670),x in [-8.000, -7.000]
S_i(x) = (0.253)x^3 + (5.152)x^2 + (34.367)x + (75.383),x in [-7.000, -6.000]
S_i(x) = (-0.244)x^3 + (-3.788)x^2 + (-19.271)x + (-31.894),x in [-6.000, -5.000]
S_i(x) = (0.095)x^3 + (1.306)x^2 + (6.197)x + (10.553),x in [-5.000, -4.000]
S_i(x) = (-0.083)x^3 + (-0.834)x^2 + (-2.364)x + (-0.861),x in [-4.000, -3.000]
S_i(x) = (-0.309)x^3 + (-2.872)x^2 + (-8.476)x + (-6.973),x in [-3.000, -2.000]
S_i(x) = (0.908)x^3 + (4.435)x^2 + (6.137)x + (2.769),x in [-2.000, -1.000]
S_i(x) = (-0.890)x^3 + (-0.960)x^2 + (0.743)x + (0.971),x in [-1.000, 0.000]
S_i(x) = (0.203)x^3 + (-0.960)x^2 + (0.743)x + (0.971),x in [0.000, 1.000]
S_i(x) = (0.445)x^3 + (-1.685)x^2 + (1.469)x + (0.729),x in [1.000, 2.000]
S_i(x) = (-0.738)x^3 + (5.414)x^2 + (-12.729)x + (10.194),x in [2.000, 3.000]
S_i(x) = (0.747)x^3 + (-7.956)x^2 + (27.380)x + (-29.915),x in [3.000, 4.000]
S_i(x) = (-0.340)x^3 + (5.092)x^2 + (-24.813)x + (39.676),x in [4.000, 5.000]
S_i(x) = (-0.112)x^3 + (1.672)x^2 + (-7.710)x + (11.170),x in [5.000, 6.000]
S_i(x) = (0.080)x^3 + (-1.777)x^2 + (12.979)x + (-30.208),x in [6.000, 7.000]
S_i(x) = (0.164)x^3 + (-3.540)x^2 + (25.326)x + (-59.016),x in [7.000, 8.000]
S_i(x) = (-0.320)x^3 + (8.066)x^2 + (-67.529)x + (188.597),x in [8.000, 9.000]
S_i(x) = (0.354)x^3 + (-10.139)x^2 + (96.324)x + (-302.962),x in [9.000, 10.000]
S_i(x) = (-0.163)x^3 + (5.393)x^2 + (-59.001)x + (214.786),x in [10.000, 11.000]
```

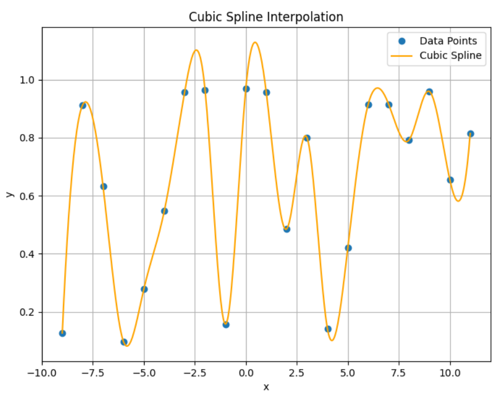
2.2 实验 (b)

```
S_i(x) = (-0.255)x^3 + (-6.875)x^2 + (-60.833)x + (-176.126),x in [-9.000, -8.000]
S_i(x) = (0.206)x^3 + (4.174)x^2 + (27.556)x + (59.577),x in [-8.000, -7.000]
S_i(x) = (0.245)x^3 + (5.003)x^2 + (33.358)x + (73.115),x in [-7.000, -6.000]
S_i(x) = (-0.217)x^3 + (-3.312)x^2 + (-16.531)x + (-26.663),x in [-6.000, -5.000]
S_i(x) = (-0.007)x^3 + (-0.163)x^2 + (-0.784)x + (-0.417),x in [-5.000, -4.000]
S_i(x) = (0.299)x^3 + (3.501)x^2 + (13.873)x + (19.126),x in [-4.000, -3.000]
S_i(x) = (-1.733)x^3 + (-14.783)x^2 + (-40.979)x + (-35.727),x in [-3.000, -2.000]
S_i(x) = (6.222)x^3 + (32.946)x^2 + (54.479)x + (27.912),x in [-2.000, -1.000]
S_i(x) = (-11.690)x^3 + (-20.789)x^2 + (0.743)x + (10.000),x in [-1.000, 0.000]
S_i(x) = (11.004)x^3 + (-20.789)x^2 + (0.743)x + (10.000),x in [0.000, 1.000]
S_i(x) = (-4.868)x^3 + (26.827)x^2 + (-46.873)x + (25.872),x in [1.000, 2.000]
S_i(x) = (0.686)x^3 + (-6.497)x^2 + (19.775)x + (-18.560),x in [2.000, 3.000]
S_i(x) = (0.366)x^3 + (-3.620)x^2 + (11.143)x + (-9.928),x in [3.000, 4.000]
S_i(x) = (-0.238)x^3 + (3.624)x^2 + (-17.832)x + (28.706),x in [4.000, 5.000]
S_i(x) = (-0.139)x^3 + (2.147)x^2 + (-10.449)x + (16.401),x in [5.000, 6.000]
S_i(x) = (0.087)x^3 + (-1.926)x^2 + (13.990)x + (-32.479),x in [6.000, 7.000]
S_i(x) = (0.162)x^3 + (-3.494)x^2 + (24.969)x + (-58.095),x in [7.000, 8.000]
S_i(x) = (-0.319)x^3 + (8.053)x^2 + (-67.408)x + (188.242),x in [8.000, 9.000]
S_i(x) = (0.354)x^3 + (-10.135)x^2 + (96.284)x + (-302.832),x in [9.000, 10.000]
S_i(x) = (-0.163)x^3 + (5.392)x^2 + (-58.991)x + (214.750),x in [10.000, 11.000]
```

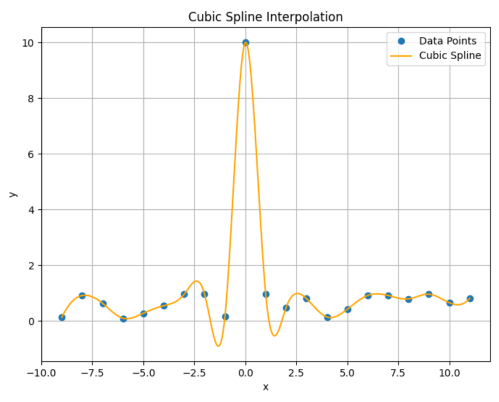
3 数据分析

通过观察程序运行结果，可以看出当 $y(0)$ 变为 10 后，函数几乎在每个区间都有变化，且变化幅度随着 x 与 0 的距离减小而增大，整体变化使得函数在整个区间上趋于光滑。为了能够让结果更加直观，我对数据做了可视化处理。

首先是用 python 的 `scipy` 库对数据求三次样条曲线，得到比较标准的图样：



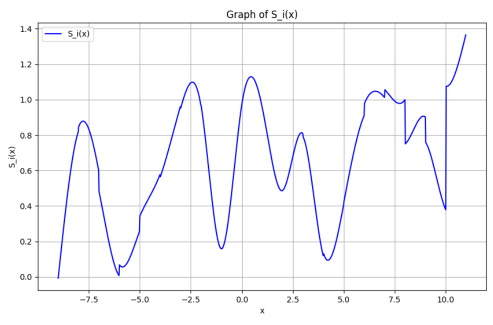
(a) $y(0) = 0.9706$



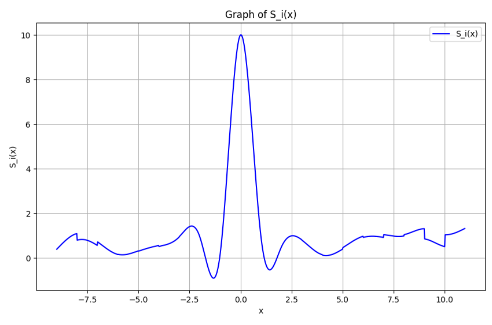
(b) $y(0) = 10$

图 1: 参考图样

接着根据我程序的运行结果画出实验图样：



(a) $y(0) = 0.9706$



(b) $y(0) = 10$

图 2: 实验图样

可以看出，实验结果符合我前面的分析，即随着 $x = 0$ 点的函数值的跳变，函数的整体取值都会发生一定的变化，其变化趋向是使得函数在整个区间上更平滑的趋向。实验图样相较于参考图样在个别极值点不够光滑原因可能是算法在数值优化方面的欠缺或者作图的方式不是最优，但总体符合预期。

4 附录：算法

Algorithm 1: Three-Spline Interpolation

```
Input: Data points array $points$ of size $ROW \times COL$ from input file
Output: Coefficients for cubic splines $S$

$h \leftarrow$ calculateH($points$);
$d \leftarrow$ calculateD($points$, $h$);
$\lambda \leftarrow$ calculateLambda($h$);
$\mu \leftarrow$ calculateMu($\lambda$);
$mat \leftarrow$ createMatrix($\lambda$, $\mu$);
$M \leftarrow$ gaussElimination($mat$, $d$);
for $i \leftarrow 0$ to $ROW - 2$ do
    $S[i][3] \leftarrow \frac{M[i+1]}{6\times h[i]} - \frac{M[i]}{6\times h[i]}$;
    $S[i][2] \leftarrow \frac{3\times points[i+1][0]\times M[i]}{6\times h[i]} - \frac{3\times points[i][0]\times M[i+1]}{6\times h[i]}$;
    $S[i][1] \leftarrow \frac{3\times points[i][0]^2\times M[i+1]}{6\times h[i]} - \frac{3\times points[i+1][0]^2\times M[i]}{6\times h[i]} - \frac{points[i][1]}{h[i]} + \frac{points[i+1][1]}{h[i]}$;
    $S[i][0] \leftarrow \frac{points[i+1][0]^3\times M[i]}{6\times h[i]} - \frac{points[i][0]^3\times M[i+1]}{6\times h[i]} + \frac{points[i+1][0]\times points[i][1]}{h[i]} - \frac{points[i][0]\times points[i+1][1]}{h[i]}$;
end
for $i \leftarrow 0$ to $ROW - 2$ do
    Print $S[i][3]$, $S[i][2]$, $S[i][1]$, $S[i][0]$ and interval $[points[i][0], points[i+1][0]]$;
end
return $S$;
```

Algorithm 2: Calculate λ

```
Input: Array $h$ containing differences between consecutive $x$ values
Output: Array $\lambda$ representing spline coefficients

Initialize array $\lambda$ of size $ROW - 1$;
for $i \leftarrow 1$ to $ROW - 2$ do
    $\lambda[i] \leftarrow \frac{h[i]}{h[i]+h[i-1]}$;
end
return $\lambda$;
```

Algorithm 3: Calculate μ

```
Input: Array $\lambda$ representing spline coefficients
Output: Array $\mu$ representing spline coefficients
1 Initialize array $\mu$ of size $ROW - 1$;
2 for $i \leftarrow 1$ to $ROW - 2$ do
3     $\mu[i] \leftarrow 1 - \lambda[i]$;
4 end
5 return $\mu$;
```

Algorithm 4: Calculate h and d

```
Input: Data points array $points$
Output: Array $h$ of differences between consecutive $x$ values and array $d$ for spline
        calculations
1 Initialize arrays $h$ and $d$ of size $ROW - 1$;
2 for $i \leftarrow 0$ to $ROW - 2$ do
3     $h[i] \leftarrow points[i+1][0] - points[i][0]$;
4     $d[i] \leftarrow \frac{6}{h[i]+h[i-1]} \left( \frac{points[i+1][1]-points[i][1]}{h[i]} - \frac{points[i][1]-points[i-1][1]}{h[i-1]} \right)$;
5 end
6 return $h$, $d$;
```